GENERATION OF UML CLASS DIAGRAM FROM SOFTWARE REQUIREMENT SPECIFICATION USING NATURAL LANGUAGE PROCESSING

Saurav Kumar Kar

110CS0074

Department of computer science and engineering,

National Institute of technology, Rourkela,

Rourkela-769008, Odisha, India.

Generation of UML Class Diagram From Software Requirement Specification Using Natural Language Processing

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE

REQUIREMENTS FOR THE DEGREE OF

Bachelor of Technology

In

Computer Science and Engineering

By

SAURAV KUMAR KAR

110CS0074

Under supervision of

Prof. Dr. S. K. RATH

Department of computer science and engineering,

National Institute of technology, Rourkela,

Rourkela-769008, Odisha, Indi

Department of Computer Science and Engineering
National Institute of Technology Rourkela
Rourkela-769008, Orissa, India.

May 12, 2014

Certificate

This is to certify that the thesis entitled Generation of UML Class Diagram from Software Requirement Specification using Natural Language Processing submitted by Saurav Kumar Kar in the partial fulfilment of the requirements for the award of Bachelor of Technology Degree in Computer Science and Engineering at National Institute of Technology, Rourkela is an authentic work carried out by him under my supervision and guidance. To the best of my knowledge, the matter embodied in the thesis has not been submitted to any other university / institution for the award of any Degree or Degree.

Prof. Dr. S. K. Rath
Professor
CSE Department of NIT Rourkela

Declaration

I hereby declare that all the work contained in this report is my own work unless otherwise acknowledged. Also, all of my work has not been previously submitted for any academic degree. All sources of quoted information have been acknowledged by means of appropriate references.

SAURAV KUMAR KAR

110CS0074

Acknowledgement

I am grateful to numerous local and global peers who have contributed towards shaping my project. At the outset, I would like to express my sincere thanks to Prof. S. K. Rath, for his advice during my project work. As my supervisor, he has constantly encouraged me to remain focused on achieving the goal. His inspiring guidance, valuable suggestion and comments helped me to establish the overall direction of the research and to move forward with investigation in depth. I would also like to thank my Ph.D. mentor Mr Abinash Tripathy. He has helped me greatly and has been a source of knowledge.

I am thankful to all my friends who have patiently extended all sorts of help for accomplishing this undertaking. I sincerely thank everyone, who has provided us with inspirational words, a welcome ear, new ideas, constructive criticism, and their invaluable time.

I would like to thank the administrative and technical staff members of the department who have been kind enough to advise and help in their respective roles. Last but not the least; I would like to dedicate this project to my family, for their love, patience and understanding.

SAURAV KUMAR KAR

110CS0074

Abbreviations

NLP: Natural Language Processing.

POS: Parts of Speech.

OOA: Object-Oriented Analysis.

UML: Unified Modelling Language.

CASE: Computer Added Software Engineering.

SRS: Software Requirements Specification.

OOAD: Object-Oriented Analysis and Design.

SVO: Subject-Verb-Object.

NN: Proper Noun

NNS: Noun, Plural

VB: Verb

RB: Adverb

DT: Determiner

MD: Modal

ABSTRACT: Any software development process starts with requirement analysis. The phase from requirement analysis to chalking out a design is acknowledged as the most intricate and troublesome exercises in programming advancement. Failures brought about throughout this action could be very unmanageable to alter in later periods of programming advancement. One primary purpose behind such potential issues is on account of the prerequisite determination being in natural language form. To conquer this, a tool has been designed, which plans can give semi-automatized aid for designers to produce UML class model from software specifications utilizing Natural Language Processing techniques. The proposed technique outlines the class diagram in a standard configuration and additionally records out the relationship between classes.

Keywords: Software Requirement Specification, UML class model, Natural Language Processing.

List of Figures

Contents

CHAPTER 1

1. INTRODUCTION

1.1 BRIEF REVIEW

Software development process is a long and tedious process. It begins with understanding the client requirements since this the backbone upon which the entire software will be modelled. This phase comprises several meetings until the final draft of requirements is prepared. This documented model of the requirements is termed as Software Requirement Specification or the SRS document.

It is this SRS which is used by the developers for building the software. It gives the information pertaining to the classes that should be present, the attributes and methods they should contain and so on. This document is human-comprehensive. But big projects have several pages of the SRS and hence practically infeasible for a human to analyse. Hence we need to design an approach to automatize this process.

These approaches have been categorised into the following approaches:

1) Traditional
2) Object oriented

The former is restricted to finding out the functions of the system only whereas the latter approach is concerned with the object oriented paradigm. It defines classes, attributes and methods. It also derives the relationship between classes if they are existent.

1.2 NATURAL LANGUAGE PROCESSING

As mentioned earlier extracting valuable information from the requirement specification document can be tedious and sometimes unpractical. It is here that we take the help of natural language processing. Our aim is to go about this analysis phase in a precise and smarter way so that we can save time. This shall also enable the developers to start with the design phase almost parallel with the analysis phase. There are several approaches to tackle this problem. However we shall adopt the most popular and widely used object-oriented approach.

1.2.1 PARTS OF SPEECH TAGGING

Parts of speech tagging, also referred to as POS tagging is the process of tagging or marking words of the input as per their figures of speech. It considers the definition as well as the context of use in the sentence while tagging any particular word. Needless to say it is also affected by the surrounding words. We usually have a database of words along with their figures of speech. But simple matching of words with those in the database won't suffice. We need to chalk out several rules that will address the effect of relative position of the word in determination of its figure of speech.

1.3 OBJECT-ORIENTED MODELLING

The entire process of software development follows the object-oriented paradigm. The models thus developed are collectively termed as UML diagrams. We have several tools to aid us in drawing these diagrams. Some of the popular ones are IBM Rational Rose, Smart Draw, UMLgraph, UMLet and many more. Using natural language for object oriented modelling was proposed almost two decades back by R.J.Abbot. Following this, several developments have been made in this field.

1.3.1 UML DIAGRAMS

UML or the Unified Modelling Language is the universally accepted standard for industrial development of software. It aids in the object-oriented approach of software modelling. UML outlines nine different diagrams which are listed below:

- Class Diagram: Depicts the system's static structure. Shows the classes with their attributes and methods. Also highlights the relation between various classes.
- Object Diagram: Closely related to class diagram. Used to validate the class diagram.
- Use-case Diagram: Shows the system's functionalities. Actors represent the entities and their functionalities are highlighted.
- Sequence Diagram: Shows the interactions between classes through message passing.
- Collaboration Diagram: Represents both static and dynamic structure of the system.
- State chart Diagram: Shows the effect of external systems on the class.

- Activity Diagram: Depicts the flow of control between activities. It is dynamic in nature.
- Component Diagram: Shows the integration of smaller components in the building of the entire system.
- Deployment Diagram: Physical and tangible system resources are depicted in this diagram.

The aforementioned models can be divided into three sub categories namely:

- Structural model
- Behavioural Model
- Architecture Model

Structural Models represents the static components in the system. It comprises the following.

- Class Diagram
- Object Diagram
- Deployment Diagram
- Component Diagram

As mentioned earlier these are the representations of the static nature of the system. Structural diagrams are not concerned with the dynamic behaviour of the system.
Class diagram is the most popular among these.

Behavioural model represents the dynamic behaviour of the system. It depicts the interactions between various elements of the system. It consists of the following:

- Activity Diagram
- Use-case diagram

Architecture model represents the entire package of the system. It thus comprises of both of the above two models. It only has one diagram:

- Package diagram

Together, these UML diagrams give the complete pictorial representation of the system to be developed. This makes it easier for the developers to build the software.

CHAPTER 2

2. LITERATURE REVIEW

2.1. NATURAL LANGUAGE PROCESSING

For the past five decades natural language processing has been a popular area of research work. Several researchers have worked on efficient and accurate retrieval of information from the SRS document. Some notable contributions in this area are Krovetz, R., & Croft, W. B [7], Salton, G., & McGill, M [8], Maron, M. E. and Kuhns, J. L [9], Losee, R. M [10]. Another milestone work in this field was done by Pedro Domingos [11] who proposed Markov Logic which could handle uncertainty and learn from the training data thus aiding in information extraction and processing. The possible usage of natural language is listed below:

1. Scanning of requirement documents.

2. Searching requirements from these documents

3. Extracting requirements from documents

4. Tagging the text for identifying many things

5. Finding similar or duplicate requirements

6. Finding probably ambiguous requirements

In a pioneering work, Ryan claims that NLP is not suitable to be used in requirement engineering (RE), as NL would not provide a reliable level of understanding, and even if it could, using such resulting system in RE is highly questionable.

2.2 OBJECT-ORIENTED APPROACH FOR SOFTWARE MODELLING

As mentioned earlier, the first pioneering work in this field was done by R.J.Abbot who proposed an approach to bridge the gap between natural language specification and object-oriented modelling. His suggestion was that the classes can be derived from the nouns in the specification and methods can be derived from the verbs. Further Buchholz highlighted that nouns not only give the classes but also the properties of the class or the attributes as we call it in object-oriented terms. Another popular proposition known as the KRB method was proposed by Kapur, Ravindra and Brown. Their proposal was to decide the classes and methods manually by inspecting the SRS document. They first identified likely candidates for classes from the nouns in the document. Similarly the class attributes were also deduced. Instantiations of classes were taken for defining the classes. Verbs were used to establish associations between classes. The attributes were normalised to map to the class to which they actually belong.

Several other works have also been done on finding out associations. These associations can be binary in nature or n-ary. Different tools were developed to reach at the UML models directly from the natural language text. A popular tool developed for this purpose is GOOAL.

2.3 UML MODEL

In relation to generate UML models from NL requirements, there have been several attempts at providing tooling support. Based on an extensive literature review, due to space limitation the following previously cited papers provide a short critique of existing tools for automatically generating UML models from NL requirements.

We divide this discussion into two categories:

1. Structural models
2. Behavioural models.

2.3.1 STRUCTURAL MODEL GENERATION

A Natural Language- Object Oriented Production System (NLOOPS) LOLITA was proposed by Mich which parses the SRS document and generates the object-oriented model. It takes help of SemNet. In this approach nouns in the document are considered as objects and the relationship among them is denoted using links. This approach cannot tackle large systems. Also it doesn't differentiate between objects and attributes.

Börstler provides a tool for constructing an object model automatically. He used keywords in the use-case description which are pre-specified. The verbs in the keywords are mapped to behaviours and nouns to objects, but require excessive user interaction to associate behaviour to the object. Nanduri and Rugaber developed a tool using syntactic knowledge by extracting objects, methods and associations and generates object diagram from NL SRS. However, these models are validated manually and user needs to have extensive domain knowledge.

CM-Builder analyses requirements texts and builds a Semantic Network, to construct an initial UML Class Model. This model can be converted into standard data interchange format, CDIF and seen using CASE tool. Human analyst can make further refinement to generate final UML models. However, CM-Builder thus generates only analysis class model.
Another tool has been developed by Popescu et.al with the aim of identifying ambiguity, inconsistency and under-specification in requirement
documents. This is done by creating object-oriented models automatically by parsing SRS according to constraining grammar. These are later diagrammed which enable human reviewer to detect ambiguities and inconsistencies.

2.3.2 BEHAVIOURAL MODEL GENERATION

There are relatively few attempts at providing tools for generating behavioural models like sequence or collaboration models from NL use-case specifications, from which design class model is generated. Li reports a semi-automatic approach to translate narrative use-case descriptions to sequence diagrams using syntactic rules and parser. He proposed eight syntactic rules to handle simple sentences which need human intervention which are insufficient to handle different types of verb phrases.

Use Case Driven Development Assistant Tool (UCDA) generates Class Model by analysing NL requirements. It aids in generating the various behavioural models like use-case diagrams, robustness diagrams and collaboration diagrams. Our approach is similar to this, but UMGAR uses accurate NLP tools in extracting models and provided design traceability mechanisms and grammatical rules for collaboration diagram generation. Main disadvantage of UCDA is that it depends on Rational Rose, a very expensive environment, for visualizing UML models. Montes et.al and Diaz et.al developed a tool to generate conceptual model, sequence diagrams, and state diagrams by analysing textual descriptions of the use-case scenarios of the system.

Yue et.al proposed a method to generate activity diagrams from use-case specifications using transformation rules. In our case, we generated collaboration diagrams from use-case specifications, as activity diagrams fails in representing which objects execute which activities, and the order in which messaging works between them. Similarly a commercial tool named Ravenflow provides mechanism to generate activity diagrams (process diagrams) from structured text written using rewriting rules. This has major limitation in representing alternative flows.

CHAPTER 3

3. PROPOSED WORK

3.1 INTRODUCTION

The proposed approach involves four major steps:

- POS Tagging
- Identifying Classes, Attributes and Methods
- Identifying relation between classes and associating attributes and methods to corresponding classes
- Pictorial representation of class diagram

3.2 POS TAGGING

This is the most important step and has already been discussed in section 1.2.1. The standard Stanford Parser has been used for this purpose. The entire document is tokenised into words. Each word is then tagged as per its figure of speech. Several rules are followed in this regard which takes into account not only the word but also its relative positioning in the sentence structure.

3.3 IDENTIFYING CLASSES ATTRIBUTES AND METHODS

Once the tagging phase is over we have at our disposal the words with their figures of speech. The tagger lists out all the words including the articles, nouns, verbs, adjectives, adverbs and so on. Now we isolate the target words. All the words listed are not of use to us. As per our assumption we need the following words:

- Nouns – Classes, Attributes
- Verbs – Methods or Operations

1) The candidates for classes are shown to the user in a window. The nouns are listed in the form of linked-list of check-boxes.
2) The user then selects the classes he needs to be developed.

3) The user selection is stored as a string and passed on to the next window.
4) The next window shows the candidates for attributes and again the user selects the desired attributes. At this stage the user is not concerned with the association of attributes with the classes. This will be incorporated later on. Again the selections are stored in a string and passed on to the next window.
5) This window has the candidates for methods. These are the verbs we have identified in our tagging phase.

3.4 RELATIONS AND ASSOCIATIONS

Once we have the list of classes, attributes and methods with us we need to identify relations between them. For this the following algorithm is used:

ALGORITHM

STEP 1: For each class scan the entire document.

STEP 2: Tokenize the document with respect to period mark (.) and thereby obtain sentences.

STEP 3: For each occurrence of the class in a sentence, search for presence of attributes or methods in that sentence.

STEP 4: List them together in a hash map.

3.5 GENERATING CLASS DIAGRAM

This is the final step of our process. The classes have already been identified along with their attributes and methods. Further the relationship between classes has also been listed out. Now this needs to be represented in a pictorial manner. We have used text-boxes in JAVA to show the classes in a manner similar to the standard format used in IBM Rational Rose.

The various classes are shown side by side and the relationship between them is showed with straight lines joining these text-boxes.

CHAPTER 4

4. IMPLEMENTATION

4.1 A CASE STUDY

The tool designed is independent of the SRS used. However to demonstrate an example we take the following Library Management System SRS as the input to the system.

"*A library can issue loan items. It caters to the needs of customers. A customer is termed as a member and is given a membership card. Each membership card has a unique member number. The customer's name, address, date of birth and other such details also need to be recorded. The library is divided into a number of sections for various subjects. Each such section has a classification mark. A loan item is distinguished by a bar code. This bar code is unique to the loan item. The loan items can be of two types namely language tapes and books. Each language tape has a title language (e.g. French), and level (e.g. beginner). Similarly a book can be identified by a title, and author(s). A customer may borrow up to a maximum of 8 items at one time. An item can be borrowed, reserved or renewed to extend a current loan. While issuing an item, the customer's membership number is scanned via a bar code reader. If membership validation succeeds and the no of loan items is less than 8, the book bar code is read, either via the bar code reader. If the item can be issued (e.g. not reserved) the item is stamped and then issued. The library also supports the facility for an item to be searched. The library database is updated on daily basis. "*

Type	Count Details
Classes	Library, Loan, Member, Customer, Book, Language, Subject, Card
Attributes	name, address, date-of-birth, bar, code, classification, title, author, Level, membership-number, valid
Methods	issue(), show(), denote(), identify(), extend(), scan(), enter(), read() , stamp(), search(). update()
Associations	Library issues Loan Items; Member Card issued to Member; Customer borrow Loan items; customer renew Loan item; customer reserve_Loan_item

Figure 1. List of Classes, Attributes, Methods and Associations

The table shows all the classes, along with their attributes and methods. It also shows the associations between classes. Here the user simply selects the classes, attributes and methods without bothering about how they are related. The tool is so designed as to incorporate the relationship between them.

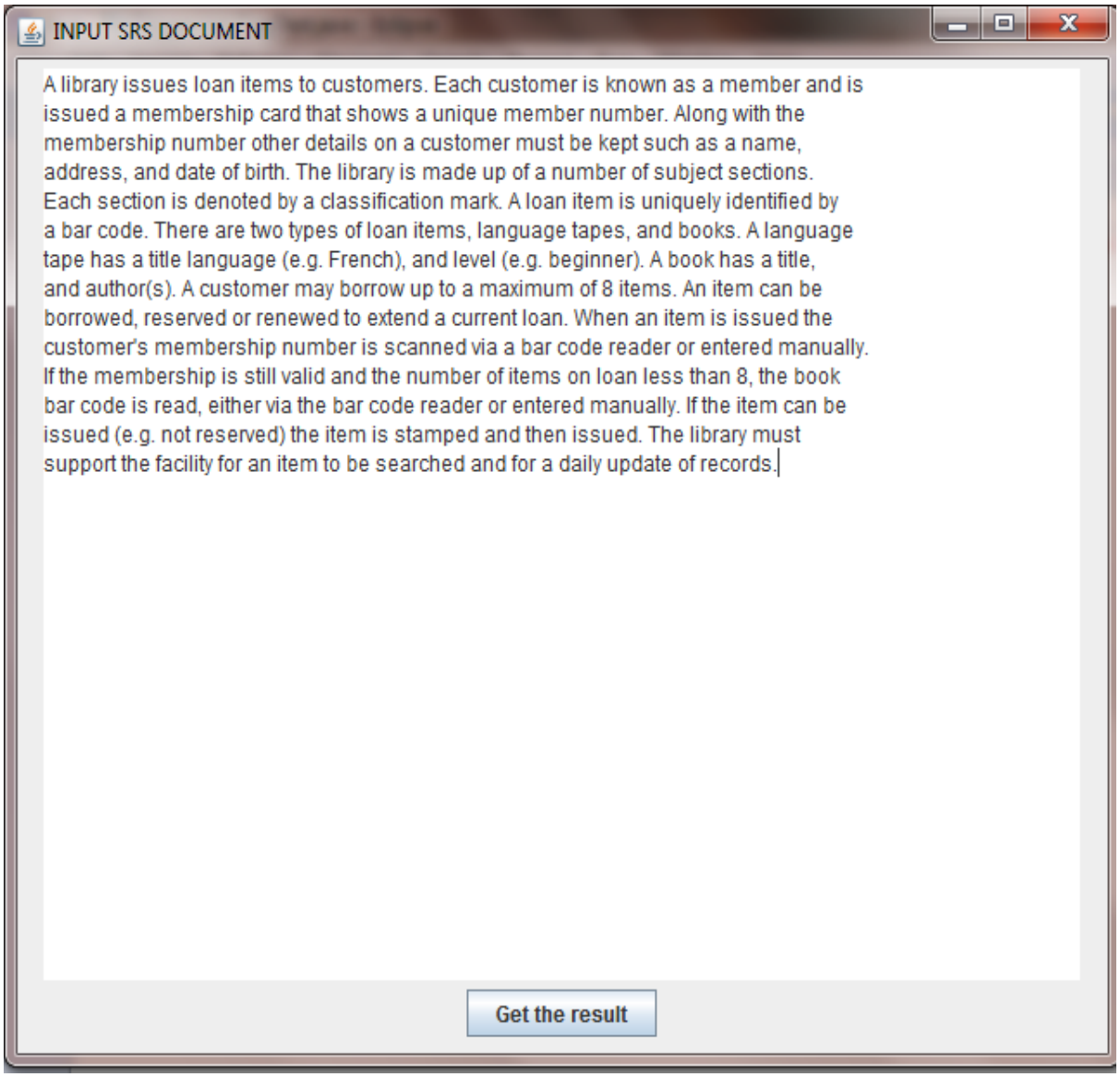

Figure 2. Input SRS Document

This is the first input window. The user is prompted to enter the SRS document for which he wants to generate the class diagram. After entering the user has to click on the "GET RESULT" button.

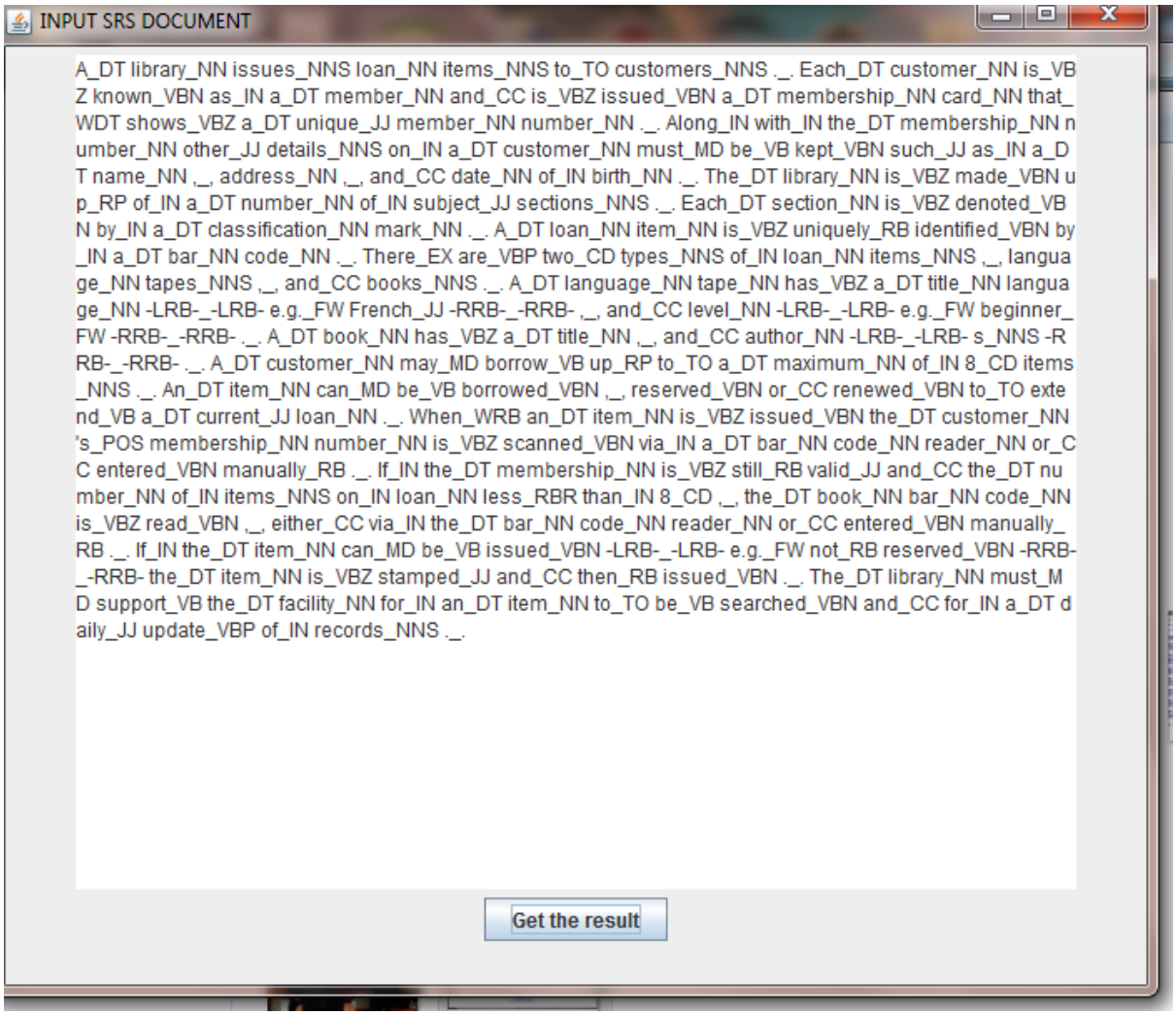

Figure 3 SRS Document after POS Tagging

The words are tagged as per their figures of speech. This is done using the standard Stanford tagger. The various abbreviations used have already been listed.

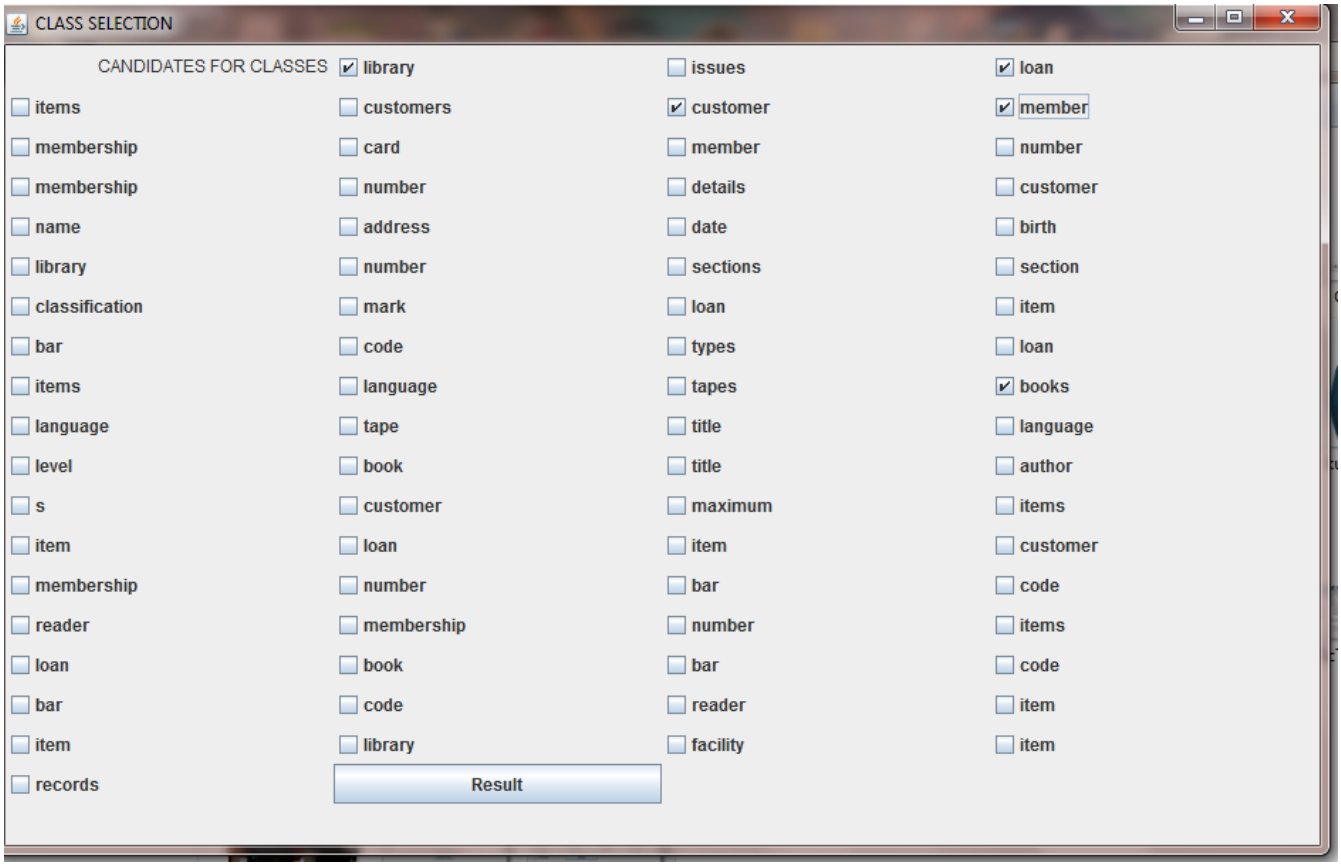

Figure 4. Class Selection

This window shows all the tentative candidates for classes as check-boxes. This is list of nouns from the tagged document. The user has to select the nouns which he requires as classes for the system. These are stored as string and passed on to the next window.

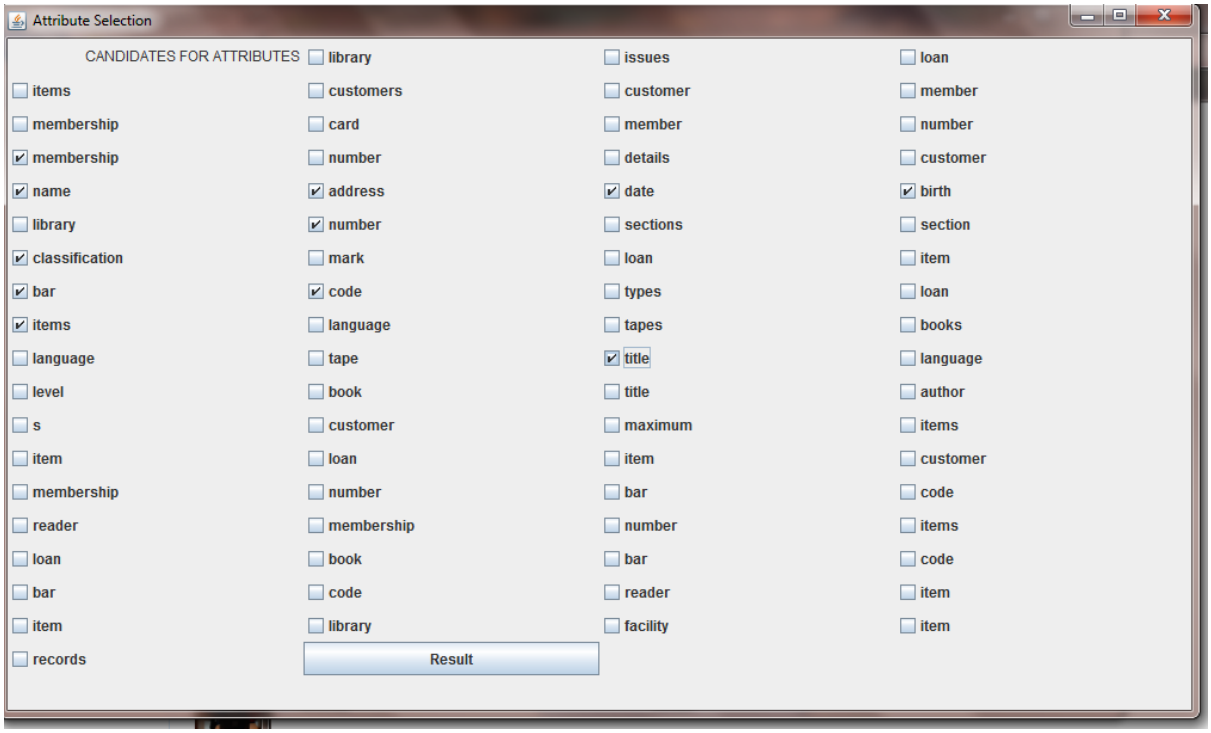

Figure 5. Attribute Selection

This figure shows the attributes list. This again is the list of nouns from the tagged document. Again the user has to select the attributes without thinking about which attribute should belong to which class.

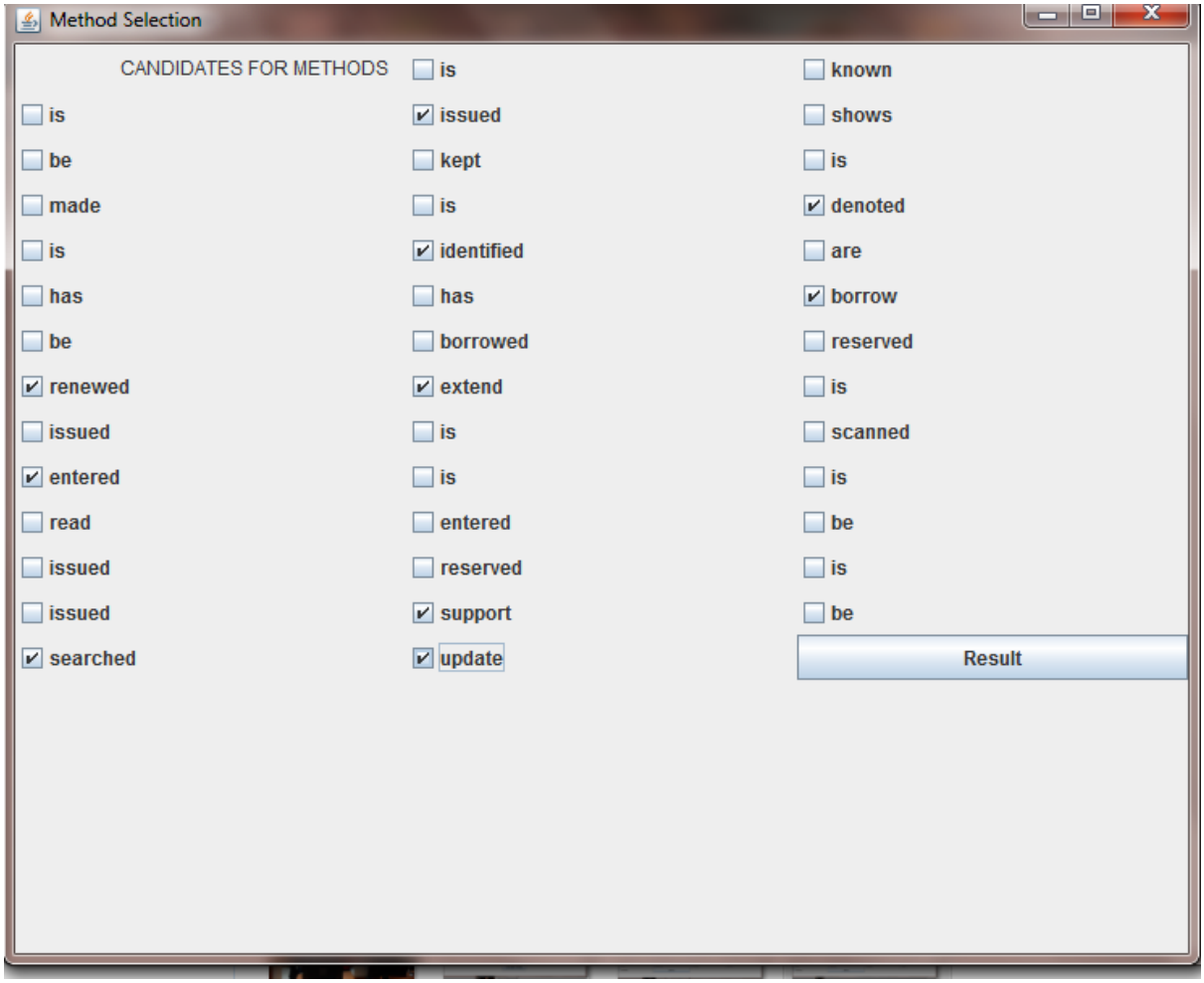

Figure 6. Method Selection

This window lists out the verbs from the tagged document and shows it as the tentative candidates for methods.

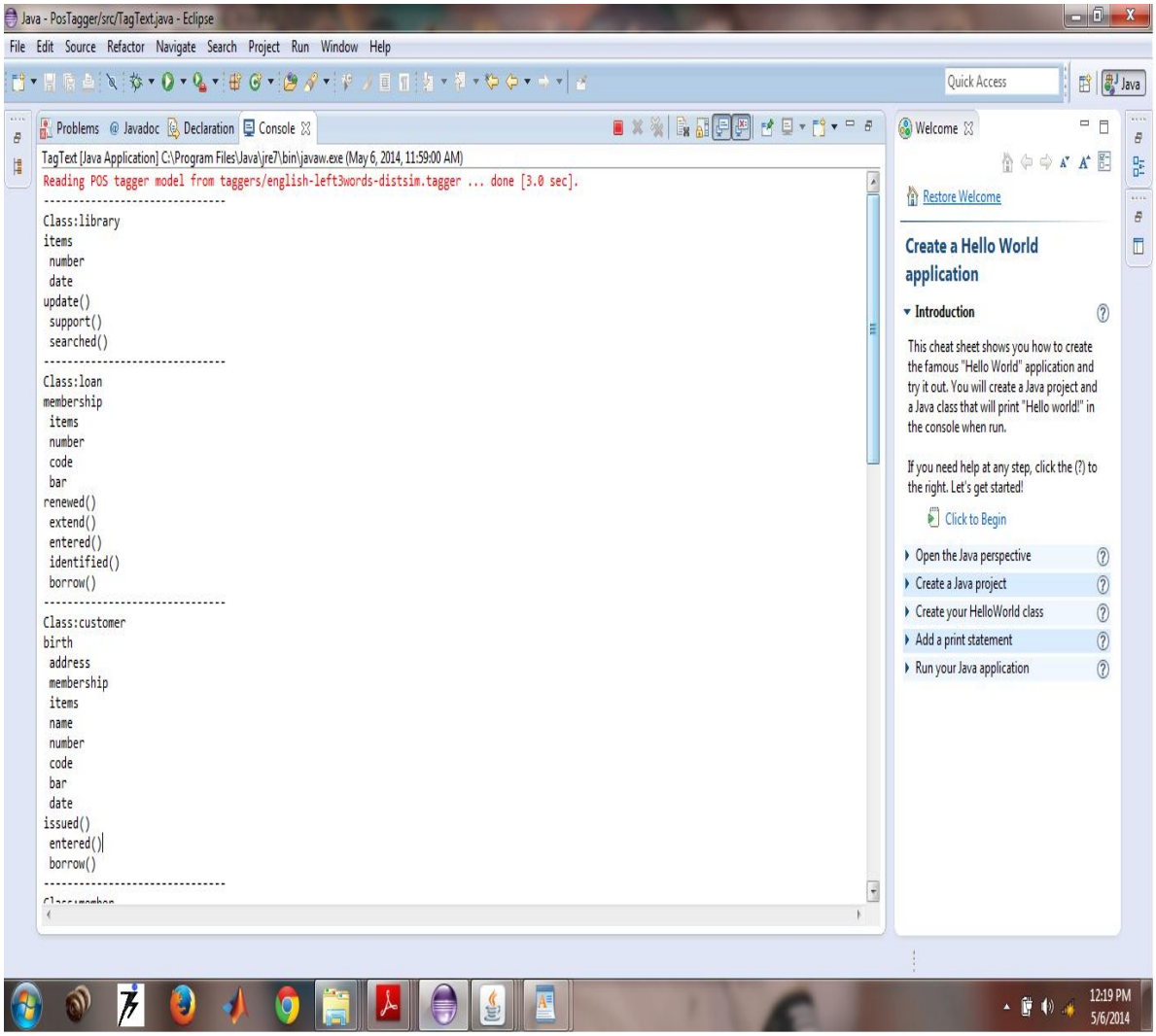

Figure 7. Association of Class with its attributes and Methods

This is the most important phase of the system. The user simply selects the candidates which he believes should be classes, attributes or methods. The task in hand is to associate attributes and methods with their corresponding classes. The algorithm proposed for this task has been stated earlier.

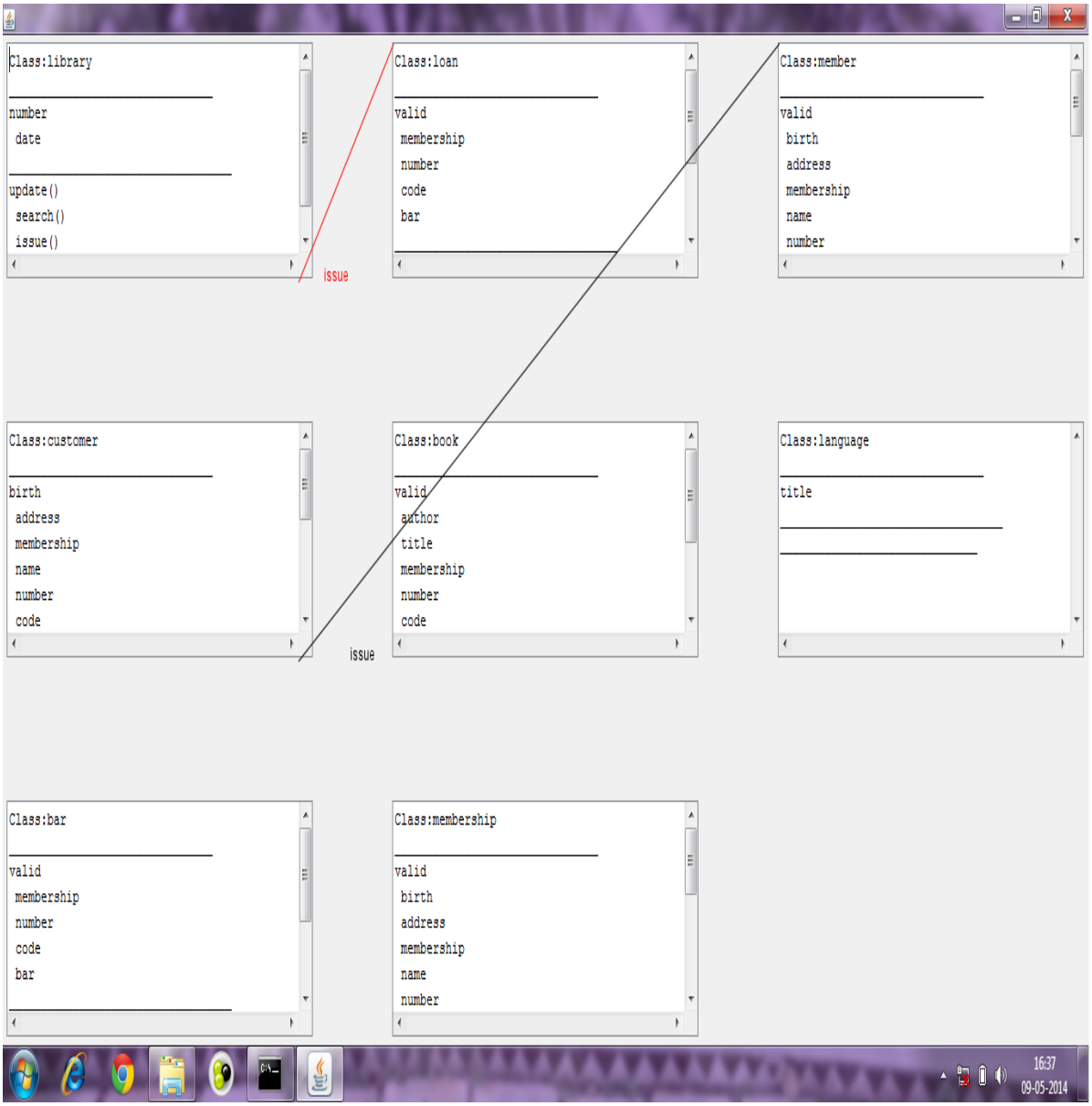

Figure 8. Class Diagram

Once the associations have been finalised, the class diagram is generated. AWT package in JAVA has been used to design this diagram. Each class is shown as a textbox with segmentations for attributes and methods. This is in accordance with the standard class diagram drawn using any standard tool like IBM Rational Rose. The associations are shown using straight lines linking the two classes and the linking relation is stated adjacently.

CHAPTER 5

5. CONCLUSION

Several attempts have been made to make the analysis of requirements an easier task. But their demit lies in the fact that each demanded careful comprehension of the specification document which in a natural language. The SRS document may be lengthy and hence stringent manual analysis is rendered impractical.

The present work is an effort to semi-automatize the generation of UML class diagram from the SRS document. This would reduce the task of the analysers involved in the software development process greatly. This would also help them estimate the cost of the project using class point approach. This is because we can readily get the no. of attributes (NOA), no. of external methods (NEM), no. of service requested etc. from the class diagram thus generated. This would greatly help the software development industry as many projects start off without any estimation of the required cost and hence end up unfinished.

The work can be further extended by:

- Automatizing the selection of classes, attributes and methods completely by considering the frequency of occurrence of words in the document.
- Identifying different modules and packages in the system.

Thus careful object-oriented analysis can go a long way in enhancing the software development process. It shall not only make it error free but also speed up the process as the developers can almost start off with the coding part once the specification is documented.

REFERENCES

[1] Ambriola V. and Gervasi V**., Processing natural language requirements.** *Proceedings of the 12th International conference on automated software engineering (ASE),* 36-45, (1997)

[2] Jackson, M. **Problems and requirements [software development]**, *Proceedings of the second international symposium on requirement engineering,* pp. 2, (1995).

[3]Bryant B.R, Lee, B.S., et al. 2008. **From Natural Language to Executable Models of Software Components**. *In Workshop on S. E. for Embedded Systems:51-58*

[4] Bajwa, Imran Sarwar, Lee, Mark G., Bordbar, Behzad. 2011. **SBVR Business Rules Generation from Natural Language Specification**. *AAAI Spring Symposium 2011, San Francisco, USA. pp.2-8*

[5] Oliveira, A., Seco N. and Gomes P. 2006. **A CBR Approach to Text to Class Diagram Translation**. *TCBR Workshop at the 8th European Conference on Case-Based Reasoning, Turkey, September 2006*

[6] Imran S. Bajwa. **From Natural Language Software Specification to UML class model**. *School of Computer Science, University of Birmingham.*

[7] Osborne, M. and MacNish, C. K. **Processing natural language software requirement specifications.** *In Proc. of 2nd IEEE Int. Conf. on Req. Engineering,* pp. 229-236, (1996)

[8] Harmain, H. M. and Gaizauskas, R. **CM-Builder: an automated NL-based CASE tool.** *In Proc. of the 15th IEEE Int. Conf. on Automated Software Engineering,* pp. 45-53, (2000)

[9] Mich, L. **NL-OOPS: from natural language to object oriented requirements using the natural language processing system LOLITA**. *Nat. Lang. Eng., 2, 2*, pp. 161-187 (1996)

[10] Overmyer, S. P., Benoit, L. and Owen, R. **Conceptual modeling through linguistic analysis using LIDA**. *In Proc. of the 23rd Int. Conf. of Software Engineering (ICSE),* Toronto, Canada,pp. 401–410, (2001)